APQ Case Study

Source: Publicaciones · **Status:** Shipped · **Period:** 2025-11 (active)

Academic publication on persisted-query behaviour, payload reduction, and operational API discipline at 90.9% cache hit rate over a 135k-line schema.

The challenge

Large GraphQL schemas pay full request-body cost on every operation even when the operation is a repeat. The cost shows up as bandwidth, CPU on the server, and p95 latency on the wire.

My role

I published a case study on Automatic Persisted Queries at scale: 90.9% cache hit rate over a 135,504-line schema, 2,089 query types, 707 Prisma models, with a cgroup v2-isolated compile runner and a five-phase auto-warmup.

What I did (3–5 decisions)

1. Two-layer cache (server-side APQ + client-side persist)

Why: Server-side APQ cuts the request body to a hash; client-side IndexedDB + Service Worker cache cuts the round trip entirely for repeat queries. The two layers cover different cases — same query, different transport.

Tradeoff: Two cache surfaces to invalidate on schema change. The schema-hash invalidation pattern handles the bulk of the problem; the edge cases need manual review.

2. Auto-warmup pre-caches the hot 2,130 queries

Why: Cold-start latency is the cost a new replica pays until the cache fills. The five-phase auto-warmup pre-caches the queries most clients will hit in the first 30 seconds, so a replica warms up before the first real client arrives.

Tradeoff: Five-minute startup ceremony per replica. Worth it for a 30-second cold-start improvement on the live traffic path.

3. cgroup v2 runner isolation for the compile step

Why: Schema compilation can OOM a shared node. The compile runner runs in a cgroup v2 slice with MemoryMax=16G so a compile spike cannot cascade into the production workers.

Tradeoff: Compile runs slower in the slice than it would on bare metal. The latency cost is bounded; the safety win is total.

What changed

Cache hit rate in production

Baseline: 0%

90.9%

Evidence: case-study-apq-performance telemetry, 2026-Q3

p95 latency reduction

Baseline: 25 ms without APQ

12 ms with APQ

Payload reduction on cached queries

Baseline: 100% body

25% body (75% reduction)

External SaaS spend for the APQ stack

Baseline: $0 — not yet implemented

$0/month — self-hosted Redis + Lua EVAL atomic counters

Overall tradeoffs

The cgroup v2 isolation adds operational surface. We chose it over a hosted compile service because the audit chain had to be self-hostable end to end.

Lessons

Two-layer caches pay back only when invalidation is honest. Schema-hash invalidation is the honest version; TTL-based invalidation is a lie that compounds over time.

Stack

GraphQL | Apollo Server v4 | Redis 7 | IndexedDB

Evidence

- [gist] APQ Case Study — English (1,336 lines, v2.1) — gist.github.com/louzt/64715cb9c6ec6ffdd98c5712b8fb7bac
- [gist] APQ Case Study — Español (1,390 lines, v2.1) — gist.github.com/louzt/0c91771bf6370f0eb47e905934ab47e8